sheng91666@163.com 's Blog (http://blog.leanote.com/sheng91666@163.com)

Love Leanote!

spring常用注解汇总

2017-02-10 11:30:33 1422 0 0

Spring注解:

使用注解前需要开启自动扫描功能，其中base-package为需要扫描的包(含子包)。

```
1. <context:component-scan base-package="cn.test"/>
```

1声明Bean注解

1	@Component	注解在类上，可以作用在任何层次。	泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注。 是一个泛化的概念，仅仅表示一个组件 (Bean)，将一个实体类，放入bean中。
2	@Service	注解在类上	用于标注业务层组件
3	@Controller	注解在类上	用于标注控制层组件（如struts中的action）
4	@Repository	注解在类上	用于标注数据访问组件，即DAO组件。

2注入Bean注解

1	@Autowired	可以对成员变量、方法、构造函数、类，进行注释。默认按类型装配	注解在set方法上或属性上 如果我们想使用按名称装配，可以结合@Qualifier注解一起使用。 如：@Autowired @Qualifier("personDaoBean") 存在多个实例配合使用
2	@Resource		默认按名称装配，当找不到与名称匹配的bean才会按类型装配。
3	@Inject	JSR-330提供的注解	

3Java配置

1	@Configuration	注解在类上	声明当前类是一个配置类，相当于一个Spring配置的xml文件。把一个类作为一个IoC容器 和@Bean搭配使用，某个方法头上如果注册了@Bean，就会作为这个Spring容器中的Bean。
2	@Bean	注解在方法上	声明当前方法的返回值为一个Bean。
3	@ComponentScan	注解在类上	自动什么包名下所有使用@Service、@Compent、@Repository、@Controller的类，并注册为Bean。
4	@Lazy(true)		表示延迟初始化
5	@Primary		自动装配时当出现多个Bean候选者时，被注解为@Primary的Bean将作为首选者，否则将抛出异常

3AOP

1	@Aspect	注解在类上	声明是一个切面。
2	@After	注解在方法上	声明建言
3	@Before	注解在方法上	声明建言
4	@Around	注解在方法上	声明建言
5	@PointCut	注解在方法上	定义拦截规则，声明切点。

4常用配置

4.1Bean的Scope

<table>
<tr><td>1</td><td>@Scope</td><td>注解在类上</td><td>用于指定scope作用域的
描述的是Spring容器如何新建Bean的实例的。实例：Scope("Prototype")
<table>
<tr><td>1</td><td>Singleton</td><td>一个spring容器中只要一个Bean的实例，此为Spring的默认配置，全容器共享一个实例。</td></tr>
<tr><td>2</td><td>Prototype</td><td>每次调用新建一个Bean实例</td></tr>
<tr><td>3</td><td>Request</td><td>Web项目中，给每一个http request新建一个Bean实例。</td></tr>
<tr><td>4</td><td>Session</td><td>Web项目中，给每一个http session新建一个Bean实例。</td></tr>
<tr><td>5</td><td>GlobalSession</td><td>这个只在portal应用中有用，给每一个 global http session 新建一个Bean实例。</td></tr>
</table></td></tr>
</table>

4.2Spring EL 和 资源调用

1	@Value	注解在变量上	调用资源（普通文件，网址，配置文件，系统环境变量等）
2	@PropertySource	注解在类上	注入配置文件是来指定文件地址。

示例：

```
//(1)增加commons-io可以简化文件相关操作
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <verson>2.3</version>
</dependency>
```

4.3Bean的初始化和销毁

1	@Bean	注解在方法上	@Bean(initMethod="init",destroyMethod="destory") 在构造函数执行后执行，在Bean销毁之前执行。
2	@PostConstruct	注解在方法上	在构造函数执行后执行
3	@PreDestroy	注解在方法上	在Bean销毁之前执行

4	@DependsOn	注解在方法上	定义Bean初始化及销毁时的顺序

4.4Profile

1	@Profile	注解在类、方法上	在不同情况下选择实例化不同的Bean。

4.5事件

4.6多线程

1	@EnableAsync	注解在配置类上	开启对异步任务的支持
2	@Async	在实际执行的Bean的方法中	声明这是一个异步任务

4.7计划任务（定时任务）

<table>
<tr><td>1</td><td>@EnableScheduling</td><td>注解在配置类上</td><td>开启对j计划任务（定时任务）的支持</td></tr>
<tr><td>2</td><td>@Scheduled</td><td>在实际执行的方法上</td><td>在要执行计划任务的方法上通过@Scheduled 声明这是一个计划任务。
包含：cron 、fixDelay、fixRate等类型
<table>
<tr><td>cron</td><td>@Scheduled（cron="0 25 11 ？ * *"）</td><td>cron是linux和unix系统下的定时任务</td></tr>
<tr><td>fixDelay</td><td>@Scheduled（fixedDelay=5000）</td><td>延迟5秒执行</td></tr>
<tr><td>fixRate</td><td>@Scheduled（fixedRate=5000）</td><td>每隔5秒执行一次</td></tr>
</table></td></tr>
</table>

4.8条件注解

1	@Conditional	用在配置类中	根据满足某一个特定条件创建一个特定的Bean。（Spring4提供）

4.9元注解

元注解：用来注解别的注解的注解。

组合注解：被注解的注解。

组合注解具备注解其上的元注解的功能。

@WiselyConfiguration=@Configuration+@CompontScan

5.0 @Enable*注解

1	@EnableAspectJAutoProxy	作用在配置类上	开启对AspectJ自动代理的支持。
2	@EnableAsync	注解在配置类上	开启对异步任务的支持
3	@EnableScheduling	注解在配置类上	开启对j计划任务（定时任务）的支持
4	@EnableWebMvc		开启web mvc的配置支持
5	@EnbaleConfigurationProperties		开启对@ConfigurationProperties注解配置Bean的支持。
6	@EnableJpaRepositories		开启对Spring Data JPA Repository的支持
7	@EnableTransactionManagement		开启注解式事务的支持
8	@EnableCaching		开启注解式的缓存支持

Spring4.2新特性：

@Order：调整配置类加载顺序。

SpringMVC注解：

1	@Controller	作用在类上	表面这个类是SpringMVC里面的Controller，将其声明为Spring的一个Bean，Dispatcher Servlet会自动扫描注解了此注解的类，并将Web请求映射到@RequestMappin的方法上。 注意： 在声明普通Bean的时候，使用@Service，@Component，@Repository，@Controller是等同的，因为@Service，@Repository，@Controller都组合了@Component。 但是在SpringMVC声明控制器Bean的时候，只能使用@Controller
2	@RequestMapping	类，方法上	用来映射Web请求(访问路径和参数)、处理类和方法的。 注解在方法上的@RequestMapping路径会继承注解在类上的路径，@RequestMapping支持Servlet的request和reponse作为参数，也支持对request和response的媒体类型进行配置。 @RequestMapping 既可以作用在类级别，也可以作用在方法级别。当它定义在类级别时，标明该控制器处理所有的请求都被映射到 /favsoft 路径下。 @RequestMapping中可以使用 method 属性标记其所接受的方法类型，如果不指定方法类型的话，可以使用 HTTP GET/POST 方法请求数据，但是一旦指定方法类型，就只能使用该类型获取数据。 @RequestMapping 可以使用 @Validated与BindingResult联合验证输入的参数，在验证通过和失败的情况下，分别返回不同的视图。 @RequestMapping支持使用URI模板访问URL。URI模板像是URL模样的字符串，由一个或多个变量名字组成，当这些变量有值的时候，它就变成了URI。
3	@ResponseBody	类，返回值前，方法上	@ResponseBody：它的作用是将返回类型直接输入到HTTP response body中。 @ResponseBody：输出JSON格式的数据

4	@RequestBody	参数前	request的参数在请求体内 1. @RequestMapping(value = "/something", method = RequestMethod.PUT) 2. public void handle(@RequestBody String body, Writer writer) throws IOException { 3. writer.write(body); 4. } 5. 6. 如果觉得@RequestBody不如@RequestParam趁手，我们可以使用 HttpMessageConverter将request的body转移到方法参数上， HttMessageConverser将 HTTP请求消息在Object对象之间互相转换，但一般情况下不会这么做。事实证明，@RequestBody在构建REST架构时，比@RequestParam有着更大的优势。
5	@PathVariable	参数前	接收路径参数。 在Spring MVC中，可以使用 @PathVariable 注解方法参数并将其绑定到URI模板变量的值上。 @PathVariable 可以有多个注解 @PathVariable中的参数可以是任意的简单类型，如int, long, Date等等。Spring会自动将其转换成合适的类型或者抛出TypeMismatchException异常。当然，我们也可以注册支持额外的数据类型。 如果@PathVariable使用Map<String, String>类型的参数时， Map会填充到所有的URI模板变量中。 @PathVariable支持使用正则表达式，这就决定了它的超强大属性，它能在路径模板中使用占位符，可以设定特定的前缀匹配，后缀匹配等自定义格式。 @PathVariable还支持矩阵变量
6	@RestController	类	组合注解 =@Controller+@ResponseBody 我们经常见到一些控制器实现了REST的API，只为服务于JSON，XML或其它自定义的类型内容。 @RestController就是这样一种类型，它避免了你重复的写@RequestMapping与@ResponseBody。 @RestController用来创建REST类型的控制器。

7	@EnableWebMvc	类	开启对SpringMVC的配置支持，可以重写这个类的方法，配置拦截器，配置静态资源映射
8	@ControllerAdvice	类	1.将对于控制器的全局配置放在同一个位置 2.注解了@Controller的类的方法可以使用@ExceptionHandler，@InitBinder，@ModelAttribute注解到方法上，这对所有注解了@RequestMapping的控制器内的方法有效。
	@ExceptionHandler		用于全局处理控制器里的异常。
	@InitBinder		用来设置WebDataBinder，WebDataBinder用来自动绑定前台请求参数到Model中。
	@ModelAttribute		本来的作用是绑定键值对到Model里，此处是让全局的@RequestMapping都能获得在此处设置的键值对。
9	@RequestParam	参数中	@RequestParam将请求的参数绑定到方法中的参数上，如下面的代码所示。 其实，即使不配置该参数，注解也会默认使用该参数。如果想自定义指定参数的话，如果将@RequestParam的 required 属性设置为 false（@RequestParam（value="id",required=false））。

<table>
<tr>
<td>10</td>
<td>@ModelAttribute</td>
<td>方法或方法参数上</td>
<td>
当它作用在方法上时，标明该方法的目的是添加一个或多个模型属性（model attributes）。该方法支持与@RequestMapping一样的参数类型，但并不能直接映射成请求。控制器中的@ModelAttribute方法会在@RequestMapping方法调用之前而调用，示例如下
<pre>
1. @ModelAttribute
2. public Account addAccount(@RequestParam String number) {
3.   return accountManager.findAccount(number);
4. }
5.
6. @ModelAttribute
7. public void populateModel(@RequestParam String number, Model model) {
8.   model.addAttribute(accountManager.findAccount(number));
9.   // add more ...
10. }
</pre>
@ModelAttribute方法用来在model中填充属性，如填充下拉列表、宠物类型或检索一个命令对象比如账户（用来在HTML表单上呈现数据）。

@ModelAttribute方法有两种风格：一种是添加隐形属性并返回它。另一种是该方法接受一个模型并添加任意数量的模型属性。用户可以根据自己的需要选择对应的风格。

当@ModelAttribute作用在方法参数上时，表明该参数可以在方法模型中检索到。如果该参数不在当前模型中，该参数先被实例化然后添加到模型中。一旦模型中有了该参数，该参数的字段应该填充所有请求参数匹配的名称中。这是Spring MVC中重要的数据绑定机制，它省去了单独解析每个表单字段的时间。

@ModelAttribute是一种很常见的从数据库中检索属性的方法，它通过@SessionAttributes使用request请求存储。在一些情况下，可以很方便的通过URI模板变量和类型转换器检索属性。
</td>
</tr>
</table>

11	@HttpEntity		HttpEntity除了能获得request请求和response响应之外，它还能访问请求和响应头，如下所示： 1. @RequestMapping("/something")public ResponseEntity<String> handle(HttpEntity<byte[]> requestEntity) throws UnsupportedEncodingException { 2. String requestHeader = requestEntity.getHeaders().getFirst("MyRequestHeader")); 3. byte[] requestBody = requestEntity.getBody(); *// do something with request header and body* 4. HttpHeaders responseHeaders = new HttpHeaders(); 5. responseHeaders.set("MyResponseHeader", "MyValue"); 6. return new ResponseEntity<String>("Hello World", responseHeaders, HttpStatus.CREATED); 7. }

springboot注解：

1	@ComponentScan(basePackages = "com.xzc.")	扫描
2	@SpringBootApplication	SpringBoot的核心注解，主要作用是开启自动配置。 @SpringBootApplication=@ComponentScan+@Conf 关闭特定的自动配置：@SpringBootApplication注解的e 例如：@SpringBootApplication(exclude = {DataSour
3	@ImportResource("classpath:ws-client.xml")	加载xml配置。
4	@EnableRedisHttpSession	
5	@Entity	注释指明这是一个实体Bean

Lombok：简化java代码注解

1	@Data	注解在类上；提供类所有属性的 getting 和 setting 方法，此外还提供了equals、canEqual、hashCode、toString 方法
2	@Setter	注解在属性上；为属性提供 setting 方法
3	@Getter	注解在属性上；为属性提供 getting 方法
4	@NoArgsConstructor	注解在类上；为类提供一个无参的构造方法
5	@AllArgsConstructor	注解在类上，为类提供一个全参的构造方法
6	@Log4j	注解在类上；为类提供一个 属性名为log 的 log4j 日志对象

----JSR-250----

1	@PostConstruct	用于指定初始化方法（用在方法上）
2	@PreDestory	用于指定销毁方法（用在方法上）
3	@Resource	默认按名称装配，当找不到与名称匹配的bean才会按类型装配。

Spring 不但支持自己定义的 @Autowired 的注释，还支持几个由 JSR-250 规范定义的注释，它们分别是 @Resource 、 @PostConstruct 以及 @PreDestroy 。

@Resource 的作用相当于 @Autowired，只不过 @Autowired 按 byType 自动注入，面 @Resource 默认按 byName 自动注入罢了。 @Resource 有两个属性是比较重要的，分别是 name 和 type，Spring 将 @Resource 注释的 name 属性解析为 Bean 的名字，而 type 属性则解析为 Bean 的类型。所以如果使用 name 属性，则使用 byName 的自动注入策略，而使用 type 属性时则使用 byType 自动注入策略。如果既不指定 name 也不指定 type 属性，这时将通过反射机制使用 byName 自动注入策略。

使用@Resource示例：

```
1. package com.baobaotao;
2.
3. import javax.annotation.Resource;
4.
5. public class Boss {
6.     // 自动注入类型为 Car 的 Bean
7.     @Resource
8.     private Car car;
9.
10.    // 自动注入 bean 名称为 office 的 Bean
11.    @Resource(name = "office")
12.    private Office office;
13. }
```

要让 JSR-250 的注释生效，除了在 Bean 类中标注这些注释外，还需要在 Spring 容器中注册一个负责处理这些注释的 BeanPostProcessor：

```
1. <bean class="org.springframework.context.annotation.CommonAnn
   otationBeanPostProcessor"/>
```

CommonAnnotationBeanPostProcessor 实现了 BeanPostProcessor 接口，它负责扫描使用了 JSR-250 注释的 Bean，并对它们进行相应的操作。

使用@PostConstruct 和 @PreDestroy：

Spring 容器中的 Bean 是有生命周期的，Spring 允许在 Bean 在初始化完成后以及 Bean 销毁前执行特定的操作，您既可以通过实现 InitializingBean/DisposableBean 接口来定制初始化之后 / 销毁之前的操作方法，也可以通过 <bean> 元素的 init-method/destroy-method 属性指定初始化之后 / 销毁之前调用的操作方法。关于 Spring 的生命周期，笔者在《精通 Spring 2.x—企业应用开发精解》第 3 章进行了详细的描述，有兴趣的读者可以查阅。

JSR-250 为初始化之后/销毁之前方法的指定定义了两个注释类，分别是 @PostConstruct 和 @PreDestroy，这两个注释只能应用于方法上。标注了 @PostConstruct 注释的方法将在类实例化后调用，而标注了 @PreDestroy 的方法将在类销毁之前调用。

示例：

```
 1. package com.baobaotao;
 2.
 3. import javax.annotation.Resource;
 4. import javax.annotation.PostConstruct;
 5. import javax.annotation.PreDestroy;
 6.
 7. public class Boss {
 8.     @Resource
 9.     private Car car;
10.
11.     @Resource(name = "office")
12.     private Office office;
13.
14.     @PostConstruct
15.     public void postConstruct1(){
16.         System.out.println("postConstruct1");
17.     }
18.
19.     @PreDestroy
20.     public void preDestroy1(){
21.         System.out.println("preDestroy1");
22.     }
23.     …
24. }
```

您只需要在方法前标注 @PostConstruct 或 @PreDestroy，这些方法就会在 Bean 初始化后或销毁之前被 Spring 容器执行了。

我们知道，不管是通过实现 InitializingBean / DisposableBean 接口，还是通过 <bean> 元素的 init-method/destroy-method 属性进行配置，都只能为 Bean 指定一个初始化 / 销毁的方法。但是使用 @PostConstruct 和 @PreDestroy 注释却可以指定多个初始化 / 销毁方法，那些被标注 @PostConstruct 或@PreDestroy 注释的方法都会在初始化 / 销毁时被执行。

使用 <context:annotation-config/> 简化配置

Spring 2.1 添加了一个新的 context 的 Schema 命名空间，该命名空间对注释驱动、属性文件引入、加载期织入等功能提供了便捷的配置。我们知道注释本身是不会做任何事情的，它仅提供元数据信息。要使元数据信息真正起作用，必须让负责处理这些元数据的处理器工作起来。

而我们前面所介绍
的 AutowiredAnnotationBeanPostProcessor 和 CommonAnnotationBeanPostProcessor 就是处理这些注释元数据的处理器。但是直接在 Spring 配置文件中定义这些 Bean 显得比较笨拙。Spring 为我们提供了一种方便的注册这些 BeanPostProcessor 的方式，这就是 <context:annotation-config/>。请看下面的配置：

示例： 调整 beans.xml 配置文件

```
1. <?xml version="1.0" encoding="UTF-8" ?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.      xmlns:context="http://www.springframework.org/schema/con
   text"
5.      xsi:schemaLocation="http://www.springframework.org/schem
   a/beans
6.  http://www.springframework.org/schema/beans/spring-beans-2.
   5.xsd
7.  http://www.springframework.org/schema/context
8.  http://www.springframework.org/schema/context/spring-context
   -2.5.xsd">
9.
10.     <context:annotation-config/>
11.
12.     <bean id="boss" class="com.baobaotao.Boss"/>
13.     <bean id="office" class="com.baobaotao.Office">
14.         <property name="officeNo" value="001"/>
15.     </bean>
16.     <bean id="car" class="com.baobaotao.Car" scope="singleto
    n">
17.         <property name="brand" value=" 红旗 CA72"/>
18.         <property name="price" value="2000"/>
19.     </bean>
20. </beans>
```

<context:annotationconfig/> 将隐式地向 Spring 容器注
册 AutowiredAnnotationBeanPostProcessor 、 CommonAnnotationBeanPostProcessor 、 PersistenceAnnotationBeanPostProcessor 以
及 equiredAnnotationBeanPostProcessor 这 4 个 BeanPostProcessor。

在配置文件中使用 context 命名空间之前，必须在 <beans> 元素中声明 context 命名空间。

使用 @Component：

虽然我们可以通过 @Autowired 或 @Resource 在 Bean 类中使用自动注入功能，但是 Bean 还是在 XML 文件中通过 <bean> 进行定义 —— 也就是说，在 XML 配置文件中定义 Bean，通过 @Autowired 或 @Resource 为 Bean 的成员变量、方法入参或构造函数入

参提供自动注入的功能。

能否也通过注释定义 Bean，从 XML 配置文件中完全移除 Bean 定义的配置呢?

答：可以的，我们通过 Spring 2.5 提供的 @Component 注释就可以达到这个目标了。

为什么 @Repository 只能标注在 DAO 类上呢?

答：这是因为该注解的作用不只是将类识别为 Bean，同时它还能将所标注的类中抛出的数据访问异常封装为 Spring 的数据访问异常类型。 Spring 本身提供了一个丰富的并且是与具体的数据访问技术无关的数据访问异常结构，用于封装不同的持久层框架抛出的异常，使得异常独立于底层的框架。

Spring 2.5 在 @Repository 的基础上增加了功能类似的额外三个注解：@Component、@Service、@Constroller，它们分别用于软件系统的不同层次：

- @Component 是一个泛化的概念，仅仅表示一个组件 (Bean)，可以作用在任何层次。
- @Service 通常作用在业务层，但是目前该功能与 @Component 相同。
- @Constroller 通常作用在控制层，但是目前该功能与 @Component 相同。

通过在类上使用 @Repository、@Component、@Service 和 @Constroller 注解，Spring 会自动创建相应的 BeanDefinition 对象，并注册到 ApplicationContext 中。这些类就成了 Spring 受管组件。这三个注解除了作用于不同软件层次的类，其使用方式与 @Repository 是完全相同的。

@Component 有一个可选的入参，用于指定 Bean 的名称，在 Boss 中，我们就将 Bean 名称定义为“boss”。一般情况下，Bean 都是 singleton 的，需要注入 Bean 的地方仅需要通过 byType 策略就可以自动注入了，所以大可不必指定 Bean 的名称。

在使用 @Component 注释后，Spring 容器必须启用类扫描机制以启用注释驱动 Bean 定义和注释驱动 Bean 自动注入的策略。Spring 2.5 对 context 命名空间进行了扩展，提供了这一功能，请看下面的配置：

```
1. 简化版的 beans.xml：
2. <?xml version="1.0" encoding="UTF-8" ?>
3. <beans xmlns="http://www.springframework.org/schema/beans"
4.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.     xmlns:context="http://www.springframework.org/schema/cont
   ext"
6.     xsi:schemaLocation="http://www.springframework.org/schem
   a/beans
7.  http://www.springframework.org/schema/beans/spring-beans-2.
   5.xsd
8.  http://www.springframework.org/schema/context
9.  http://www.springframework.org/schema/context/spring-context
   -2.5.xsd">
10.     <context:component-scan base-package="com.baobaotao"/>
11. </beans>
12.
13. 这里，所有通过 <bean> 元素定义 Bean 的配置内容已经被移除，
14. 仅需要添加一行 <context:component-scan/> 配置就解决所有问题了--Spr
   ing XML
15. 配置文件得到了极致的简化（当然配置元数据还是需要的，只不过以注释形式存在
   罢了）。<context:component-scan/> 的 base-package： 属性指定了需
   要扫描的类包，类包及其递归子包中所有的类都会被处理。
```

<context:component-scan/> 还允许定义过滤器将基包下的某些类纳入或排除。Spring 支持以下 4 种类型的过滤方式，通过下表说明：

过滤器类型	说明
注释	假如 com.baobaotao.SomeAnnotation 是一个注释类，我们可以将使用该注释的类过滤出来。
类名指定	通过全限定类名进行过滤，如您可以指定将 com.baobaotao.Boss 纳入扫描，而将 com.baobaotao.Car 排除在外。
正则表达式	通过正则表达式定义过滤的类，如下所示：com\.baobaotao\.Default.*
AspectJ 表达式	通过 AspectJ 表达式定义过滤的类，如下所示： com.baobaotao..*Service+

示例：

```
<context:component-scan base-package="com.baobaotao">
    <context:include-filter type="regex"
        expression="com\.baobaotao\.service\..*"/>
    <context:exclude-filter type="aspectj"
        expression="com.baobaotao.util..*"/>
</context:component-scan>
```

值得注意的是 <context:component-scan/> 配置项不但启用了对类包进行扫描以实施注释驱动 Bean 定义的功能，同时还启用了注释驱动自动注入的功能（即还隐式地在内部注册了 AutowiredAnnotationBeanPostProcessor 和 CommonAnnotationBeanPostProcessor），因此当使用 <context:component-scan/> 后，就可以将 <context:annotation-config/> 移除了。

默认情况下通过 @Component 定义的 Bean 都是 singleton 的，如果需要使用其它作用范围的 Bean，可以通过 @Scope 注释来达到目标，如以下代码所示：

使用@Scope：

```
通过 @Scope 指定 Bean 的作用范围
package com.baobaotao;
import org.springframework.context.annotation.Scope;
…
@Scope("prototype")
@Component("boss")
public class Boss {
    …
}
```

这样，当从 Spring 容器中获取 boss Bean 时，每次返回的都是新的实例了。

上一篇: 封装、继承、多态
(http://blog.leanote.com/post/sheng91666@163.com/%E5%B0%81%E8%A3%85%E3%80%81%E7%BB%A7%E6%89%BF%E3%80%

下一篇: Mybaties注解
(http://blog.leanote.com/post/sheng91666@163.com/Mybaties%E6%B3%A8%E8%A7%A3)

导航

主页 (http://blog.leanote.com/sheng91666@163.com)
About Me (http://blog.leanote.com/single/sheng91666@163.com/About-Me)
归档 (http://blog.leanote.com/archives/sheng91666@163.com)
标签 (http://blog.leanote.com/tags/sheng91666@163.com)

最近发表

MySQL、Oracle数据类型比较
(http://blog.leanote.com/post/sheng91666@163.com/MySQL%E3%80%81Oracle%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B%E6%AF%94
单例设计模式
(http://blog.leanote.com/post/sheng91666@163.com/%E5%8D%95%E4%BE%8B%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F)
集合 (http://blog.leanote.com/post/sheng91666@163.com/%E9%9B%86%E5%90%88)
详解变量、关键字、代码块 (http://blog.leanote.com/post/sheng91666@163.com/%E5%8F%98%E9%87%8F)
Date类 (http://blog.leanote.com/post/sheng91666@163.com/Date%E7%B1%BB-2)

友情链接

My Note (https://leanote.com/note)
Leanote Home (https://leanote.com)
Leanote BBS (http://bbs.leanote.com)
Leanote Github (https://github.com/leanote/leanote)